TN1205.01
Technical Note

USING THE XE1205 FIFO

Table of Contents

1 Introduction. **3**
2 Signal Path In the XE1205 Chip. **3**
2.1 Demodulation Process. 3
2.2 FIFO filling. 4
2.3 Serial Control Interface 5
3 Problems with the SPI – FIFO interface **7**
3.1 Symptoms 7
3.2 Explanation. 8
4 Workaround **10**

1 INTRODUCTION.

The XE1205 products whose date-codes start with N4K and N5K (except N5K3760 and N5K3760A) exhibit a non-conformance to specification. The non-conformance affects the FIFO buffer described in section 5.2.4 of the XE1205 data sheet. This present document describes the normal behavior of the XE1205 built-in FIFO used in RX mode (interaction between bit synchronizer, pattern matching, FIFO and SPI). It also describes and proposes a workaround to the problem encountered using the FIFO buffer of the N4K and N5K (except N5K3760 and N5K3060A) versions of the XE1205. All other date-codes are in conformance with the specification.

2 SIGNAL PATH IN THE XE1205 CHIP.

2.1 Demodulation Process.

The XE1205 is a direct conversion (Zero-IF) half-duplex data transceiver. It includes receiver, transmitter, and frequency synthesizer and control logic. The circuit is intended for operation in the following three frequency bands 433 MHz, 868 MHz, and 915 MHz and uses 2-level FSK modulation.
The receiver converts the incoming 2-level FSK modulated signal into a synchronized bit stream. The receiver comprises a low-noise amplifier, down-conversion mixers, baseband filters, baseband amplifiers, limiters, demodulator and bit synchronizer. The bit synchronizer transforms the data output of the demodulator into a glitch-free bit stream DATAOUT and synchronized clock DCLK.

The XE1205 is user-programmable between two modes of operation:
Continuous mode: each bit transmitted or received is accessed directly at the DATA input/output pin.
Buffered mode: a 16-byte FIFO is used to store each data byte transmitted or received. This data is written to/read from the FIFO via the SPI bus.
In this mode, the output of the bit synchronizer, i.e. the demodulated and resynchronized signal and the clock signal DCLK are not sent directly to the output pins DATA and IRQ_1 (DCLK). These signals are used to store the demodulated signal by packet of 8 bits in a 16 bytes FIFO. The following figure shows the receiver chain in this mode. More information about the XE1205 and registers settings can be found in the XE1205 datasheet.

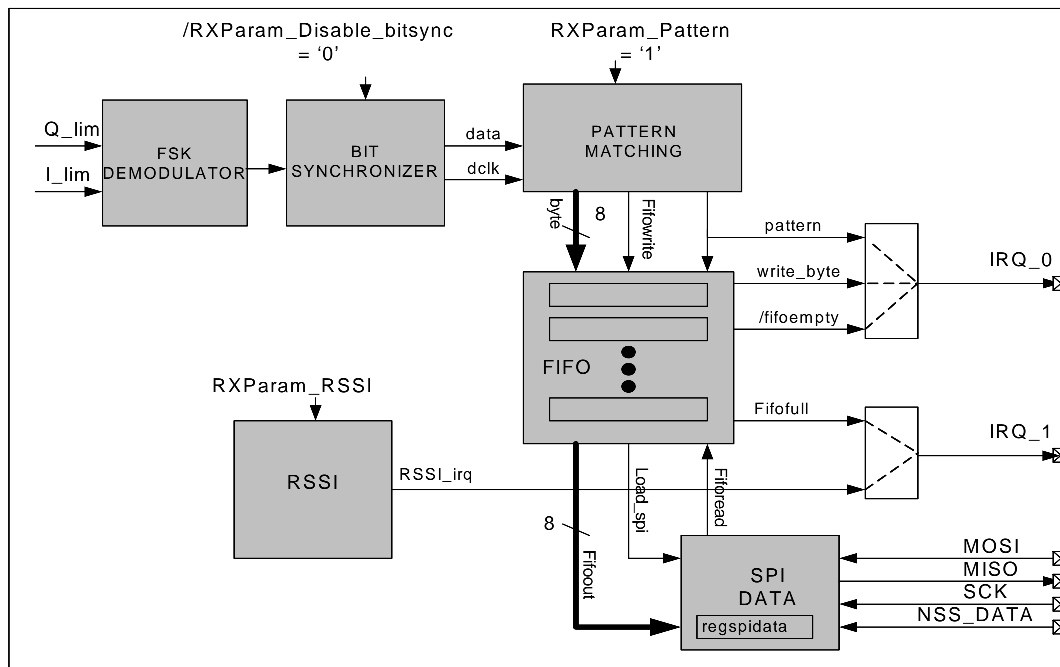

Figure 1: Receiver chain in buffered mode

2.2 FIFO Filling.

When the chip is in receive mode and the MCParam_Buffered_mode bit is set to high then all the blocks described above are enabled. In a normal communication frame the data stream comprises a 24 bit preamble, pattern and the data. Upon receipt of a recognized pattern, the receiver recognizes the start of a frame, strips off the preamble and pattern, then transmits the data to the microcontroller. This automated data recovery reduces the overhead for the host controller.
The following figure shows the standard FIFO filling.

The IRQParam_Start_fill bit determines how the FIFO is filled:

If IRQParam_Start_fill is low, data only fills the FIFO subject to a correct pattern match. Data is shifted into the pattern recognition block which continuously compares the received data with the contents of the Reg_pattern(31:0) configuration register. If a match occurs a start sequence is detected, and the internal output of the pattern matching block is asserted for one bit length and the IRQParam_Start_detect bit is also asserted. This internal signal may be mapped to the IRQ_0 output using interrupt signal mapping. Once a pattern match has occurred, the pattern recognition block will remain inactive until IRQParam_Start_detect is re-asserted.

If IRQParam_Start_fill is high, FIFO filling is initiated by asserting IRQParam_Start_detect.

Once sixteen bytes have been written to the FIFO the IRQParam_Fifofull signal is asserted. Data should then normally be read out. If no action is taken the FIFO will overflow and subsequent data will be lost. If this occurs the IRQParam_Fifooverrun bit is set. The IRQParam_Fifofull signal can be mapped to pin IRQ_1 as an interrupt for a microcontroller if IRQParam_RX_irq_1 is set to “01”

To recover from an overflow situation a ‘1’ must be written to IRQParam_Fifooverrun; this clears the contents of the FIFO, resets all FIFO status flags and re-initiates pattern matching.

Pattern matching can also be re-initiated during a FIFO filling sequence by writing a ‘1’ to IRQParam_Start_detect.

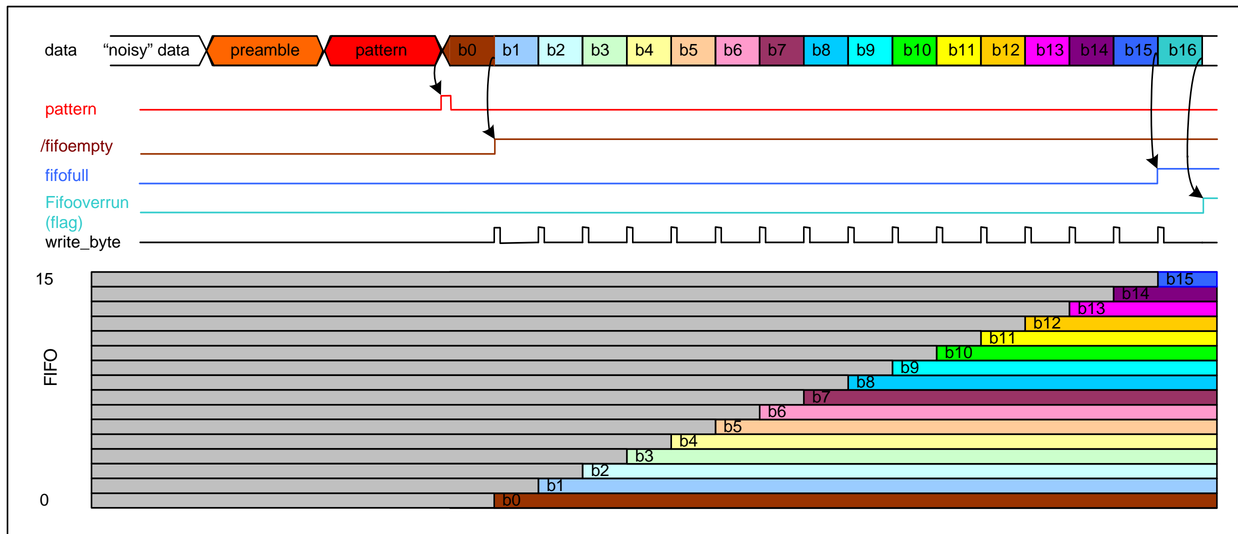

Figure 2: Standard FIFO filling

The FIFO filling process is shown in detail in Figure 2. As the first byte is written into the FIFO the signal /fifoempty goes high indicating that at least one byte is present. The microcontroller can then read the contents of the FIFO via the SPI interface. Once all data have been read from the FIFO then /fifoempty goes low. Once the last bit of the sixteenth byte has been written into the FIFO then the signal Fifofull is asserted; data should be read before the next byte is received. This is described in Figure 2 .

The /fifoempty signal can be used as an interrupt signal for a microcontroller by mapping to pin IRQ_0 if IRQParam_RX_irq_0(1:0) is set to “10”. Alternatively, the WRITE_BYTE signal may also be used as an interrupt if IRQParam_RX_ireq_0(1:0) is set to “01”.

© Semtech 2006

2.3 Serial Control Interface

The XE1205 contains two SPI-compatible serial interfaces, one to send and read the chip configuration, the other to send and receive data in buffered mode. Both interfaces are configured in slave mode and share the same pins MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock). Two additional pins are required to select the SPI interface: NSS_config to change or read the transceiver configuration, and NSS_data to send or read data.

Figure 3 shows the connections between the transceiver and a microcontroller when buffered mode is used.

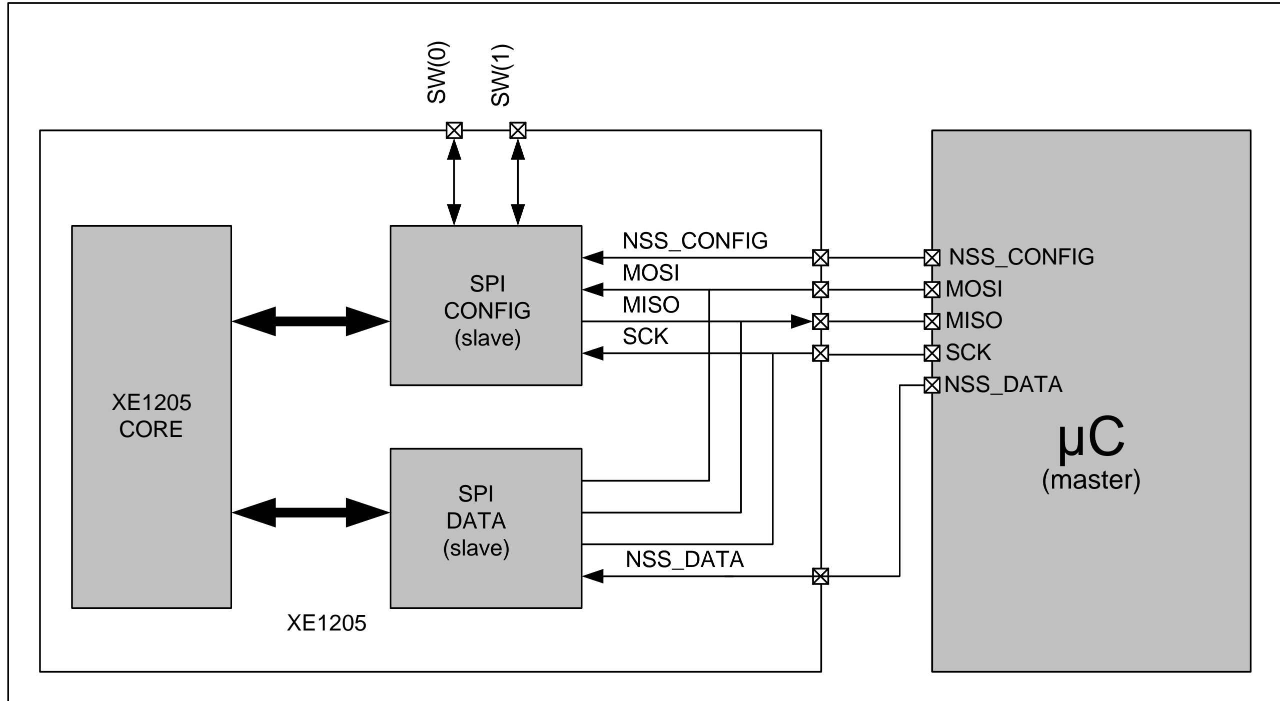

Figure 3: Connection between SPI DATA, SPI CONFIG and a micro-controller

By default, the serial control interface is used for configuration. It is also possible to change between the four modes (sleep, stand-by, receive, transmit) by using the two-bit signal SW(1:0). This option is enabled by setting the bit MCParam_Select_mode to ‘1’ in the configuration register.

A byte transmission can be seen as a rotate operation between the value stored in an 8 bit shift register of the master device (the microcontroller for instance) and the value stored in an 8 bit shift register of the selected slave device (the transceiver). The SCK line is used to synchronize both SPI interfaces. Data is transferred full-duplex from master to slave through the MOSI line and from slave to master through the MISO line. The most significant bit is always sent first. In both SPI interfaces the rising SCK edge is used to sample the received bit, and the falling SCK edge shifts the data inside the shift register.

The NSS_config or NSS_data signal is controlled by the master device and should remain low during the byte transmission. It is not necessary to toggle the NSS_data or NSS_config signal back to high and back to low between each transmitted byte. The transmission is synchronized by the NSS_config or NSS_data signal. While the NSS_config or NSS_data is high, the counters controlling transmission are reset. Reception starts with the first clock cycle after the falling edge of NSS_config or NSS_data; if either signal goes high during a byte transmission the counters are reset and the byte has to be retransmitted. When the transceiver is used in buffered mode, the data exchange with a micro-controller is via the SPI_DATA interface.

In transmit mode the16 byte FIFO is filled whilst the interrupt signal IRQ_1 (TX_FIFOfull) is low.

© Semtech 2006

In receive mode, the FIFO may be read if one of the following events occurs:

- at least one byte is present in the FIFO, i.e. a rising edge on IRQ_0 mapped to /fifoempty
- each time a byte is written to FIFO, i.e. a rising edge on IRQ_0 mapped to WRITE_BYTE
- 16 bytes have been written to the FIFO, i.e. a rising edge on IRQ_1 mapped to RX_FIFOfull

The transceiver should be in buffered mode (MCParam_Buffered_mode = ‘1’). The SPI_DATA interface is then selected if NSS_data is low and NSS_config is high.

The operations with SPI_DATA interface are similar to those with SPI_CONFIG except that there is only a data byte (no address byte is required).
The following figure shows the read operation in receive mode:

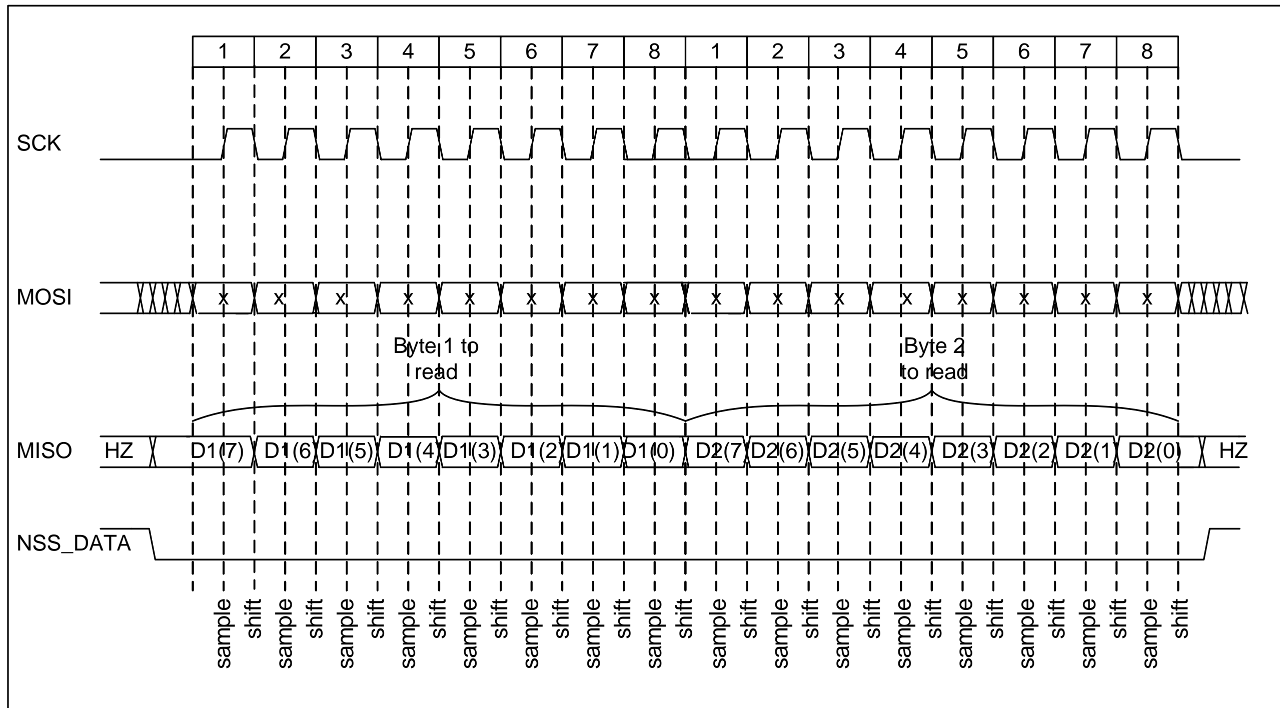

Figure 4: Reading 2 bytes in receive mode.

3 PROBLEMS WITH THE SPI – FIFO INTERFACE.

It has been noticed that despite the fact that the FIFO filling process works well on the current version of the chip, problems appear when using the serial interface.

3.1 Symptom

The problem appears when the chip is used in RX buffered mode. A data frame is sent by a transmitter and demodulated using the XE1205. As explained in chapter 2, when a preamble is sent, followed by a known pattern and if IRQParam_Start_fill is low, the FIFO is filled with the data byte per byte and the user can read those bytes using the SPI.

They are several ways to read the FIFO content:

- One can choose to wait until FIFO is full and then read the FIFO via the SPI until the FIFO is not empty, wait again for fifofull and so on until the end of frame is detected.
- One can also decide to wait until an overrun occurs and read the FIFO until the FIFO is empty. This implies that the overrun flag will be clear and a new start of frame sent to restart the process. This is especially useful when the frame lengths are smaller that 16 bytes.
- One can choose to start to read the FIFO when at least on byte is present in the FIFO (indicated by /fifoemty flag). No read operation occurs when the /fifoempty flag is low.
- Once several bytes are present but the FIFO is not full (the Write_byte signal allows counting the number of byte in the FIFO). The read operation is done until FIFO is empty, then the application start to count again the number of byte present and so on until the end of frame is detected.

For each case described above the following algorithm is used:

```
While /FIFOempty = ‘1’ do
{
read_fifo();
}
```

When such algorithm is used, the user will loose bytes and especially the last byte present into the FIFO.

© Semtech 2006

3.2 Explanation.

When the SPI is used to read the FIFO, the data on the output of the FIFO is put into the SPI buffer and sent as describe in figure 5. In the same time the FIFO output is updated with the next value.

They are two reasons for the problem observed:

1) When the first byte is stored in the FIFO, this byte also forces the SPI buffer in the same time so that the data is ready to be read upon request of the microcontroller. This is done automatically even if the SPI is being used or not. This is illustrated in the following figure (point 1)

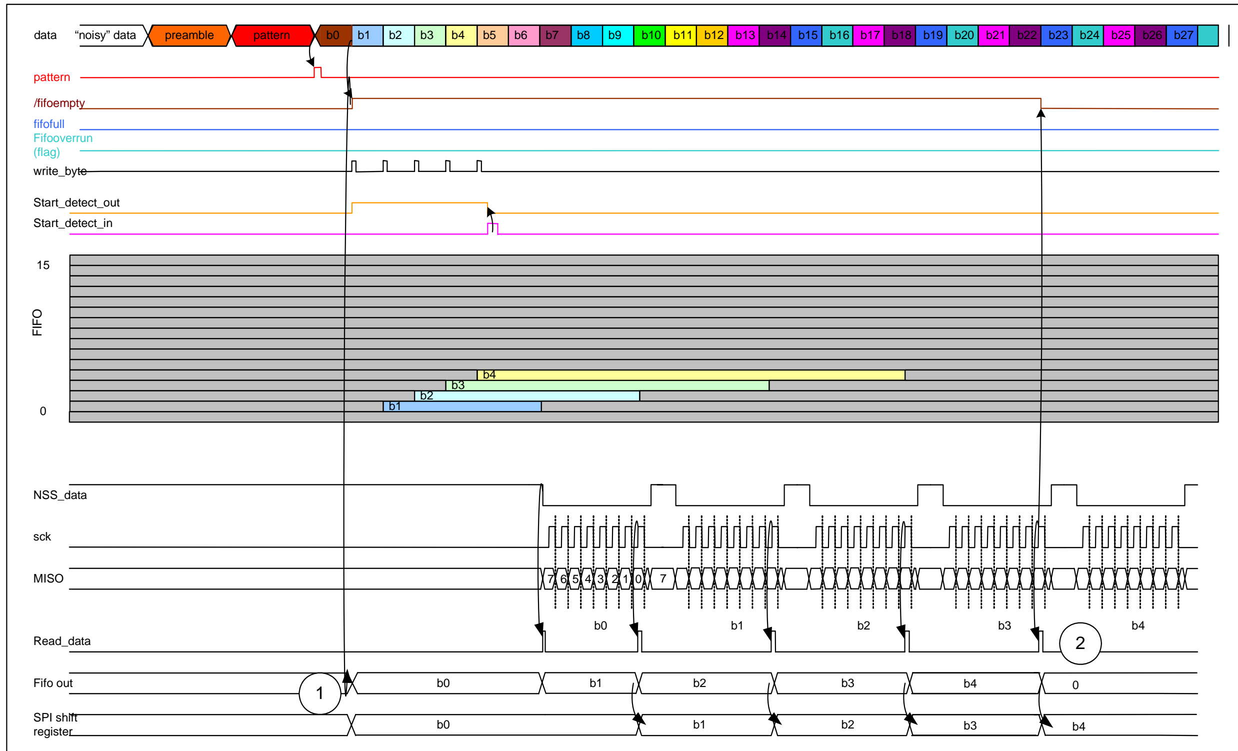

Figure 5: Standard RX timing

FFIO out represents the byte currently at the FIFO output. SPI shift register represent the SPI buffer.

2) Since It is not necessary to toggle the NSS_data or NSS_config signal back to high and back to low between each transmitted byte and since the data are sampled on the first rising edge of the serial clock SCK, the next byte is loaded into the SPI buffer at the end of the current byte transfer. The FIFIO is read in the same time. This is illustrated in the figures above (point 2).

This implies that the FIFO flag becomes empty not during the transfer of the last byte present in the FIFO but at the end of the previous one. If the algorithm described above is applied then the last byte is never read. The /fifoempty flag comes fall, but on more SPI access is needed to read the byte present in the SPI buffer. In this situation, the FIFO flag is empty, data is present in the SPI buffer but not read. If no data is written into the FIFO then one more SPI access will allow reading the last byte and no problem appears. If data is written when /fifoempty flag is low, then the SPI buffer is overwritten automatically with the new byte and /fifoempty flag goes high. If this occurs during an SPI access (needed to read the last byte) then the data is corrupted. If this occurs when no SPI access is running then the byte present into the SPI buffer is overwritten and lost. The user will be able to read the FIFO again since the FIFO is not empty but one byte is lost. If one continues the operation the last byte read before /fifoempty flag becomes low will also be lost and so on. The following example shows a situation where after receiving 4 bytes in the FIFO, one starts to read the FIFO until it is empty. A new byte is written to the FIFO during the extra SPI access needed. In that case the data send via SPI is corrupted.

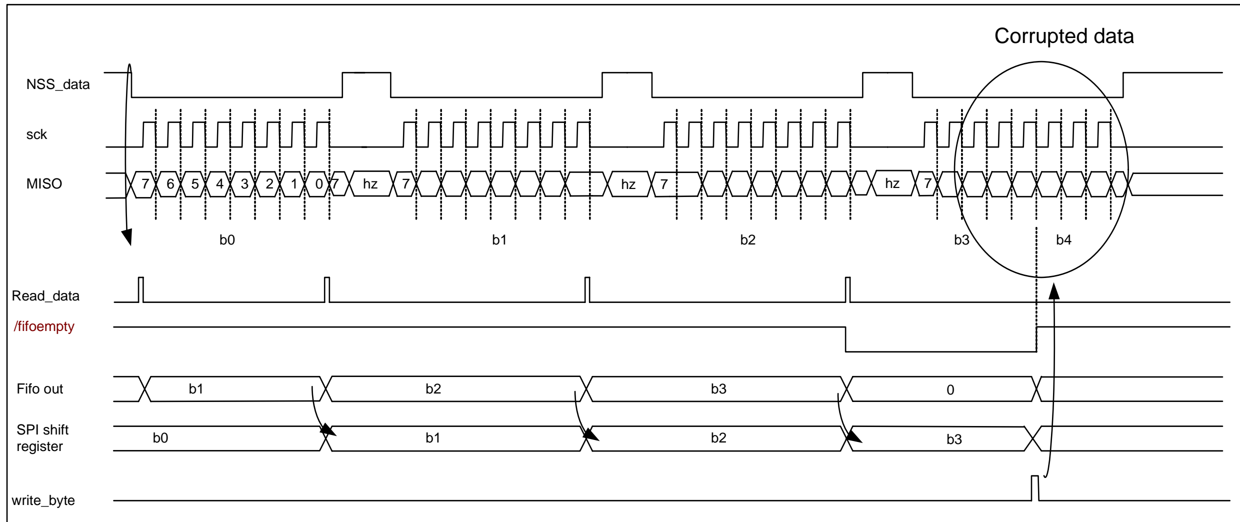

Figure 6: Corruptions of data

The following figure shows the case where a byte is lost. A byte is written into the FIFO while the last byte is present in the SPI buffer but not read yet. The buffer is overwritten and the next SPI access will send the next byte to the microcontroller i.e. the current byte is lost.

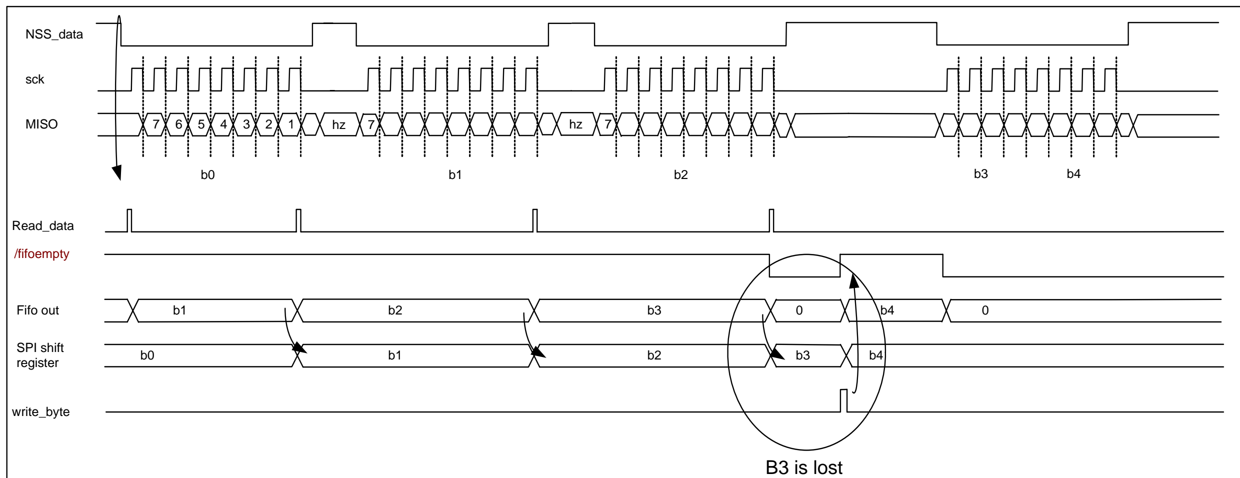

Figure 7: Losses of data

© Semtech 2006

4 WORKAROUND

Based on this observation, a workaround has been found. The main idea of this workaround is that the SPI buffer must not be overwritten by the FIFO which means that the /fifoempty flag must remain high in any case. The problem occurs only when the FIFO is empty (/fifoempty low). The solution proposed is to wait until the FIFO is full, and then read via SPI 14 bytes and so on. If one does not wait until the FIFO is full then you must ensure that there will be least one 2 bytes left in the FIFO. The reason two bytes and not one is because the /fifoempty flag comes back to low at the end of the read process of the byte before the last one. Doing this will avoid having the FIFO empty and loosing bytes. In addition, **it is now necessary to toggle the NSS_data or NSS_config signal back to high and back to low between each transmitted byte.**

An example of C code is:

```
While ((RegPAIn & 0x02) == 0x00) {} // Wait for FIFOFULL (IRQ0/PA1) signal
            for (i=14 ; i!=0 ; i-- ) {                    // Read 14 bytes from FIFO
    RegPBOut = ReceiveByte();           // and transfer them to PortB
    }
```

© Semtech 2006
All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804